# The Scale Filter for IconBuilder:

IconBuilder tools and filters are just special bundles. The bundles contain executable code for the tool, the interface for the tool's inspector, and any accessory resources that that the tool needs to function (icons, sounds, language specific strings, etc.). Making a new tool is surprisingly easy, and is accomplished through the magic of Objective-C and NXBundle objects. If you want to know more about the design IconBuilder, take a look at "An introduction to extensible programming" by Jeff Martin and Joshua Doenias from the Fall 1991 Next On Campus. In that article, the authors describe an extensible painting program called "DynaPaint" which bears a striking resemblance to IconBuilder!

The IconBuilderScaleFilter directory should contain the following items:

- README.rtf (this file)
- Scale.pfilter (a compiled and ready-to-go filter for scaling images)

- Scale (a ProjectBuilder directory with the filter's source code)

**Using the Scale filter:**

Start up IconBuilder and load Scale.pfilter by selecting Tools -> LoadTool... from the menu. Alternately, Scale.pfilter will be automatically loaded when IconBuilder is launched if it is placed in IconBuilder's app wrapper. To scale an image, drag out a rectangle with the selection tool, bring up the tool inspector, drag the pull-down menu to Scale, change any parameters if necessary, and click Apply.

The scaling algorithm parameters are:

- Scale Factor -- The scale factor controls how much bigger or smaller to make the image. A scale factor of 2 will make the image twice as big, while a scale factor of .5 will shrink the image to half its original size .

- Focus -- The focus determines how sharp the output image will be. A focus of 0 will create an image with very sharp boundaries, a focus of 1 smooths out the boundaries of the image by about the width of a pixel or so, and a focus of 10 will create very blurry images. The optimal value for the focus parameter varies from image to image, but values in the range of .7 to 2 usually produce good results.

- Samples -- The samples parameter controls how many samples of the input image will be taken and averaged when producing each output pixel. Instead of using the normal convolution method for scaling images, Scale.pfilter uses a stochastic sampling method to achieve the same result. As the number of samples increase, the output image will become smoother. The useful upper limit on the number samples is 256.

The time it takes to scale an image depends on the number of pixels in the output image and the number of samples to take per pixel. Since scaling

can be quite slow for large images or a large number of samples, a progress monitor is displayed in the inspector panel during the scaling operation.

Warning #1: The scaling algorithm is non-deterministic. This means that it is very unlikely you will get exactly the same result twice (unless the samples parameter is set very high). There is almost never any perceptible difference, but many people will need something that is totally predictable. Those people should not use this filter.

Warning #2: The scaling algorithm is very slow. Speed is not a problem when producing output images that are icon-size, but creating large output images can be interminably slow.

**The Progress View:**

The ProgressView class is an example of how the scale filter can be used to create anti-aliased images. If you are starved for memory , just

remove ProgressDials.tiff from Scale.pfilter and the dial will be rendered from normal postscript operators. The anit-aliased dials will also not be used on machines with only a two bit framebuffer.

**Compiling The Scale Filter:**

It appears that ProjectBuilder does not let you get at the CFLAGS macro in the standard makefiles, so you are stuck with the default -g -O -Wall. In addition, strip doesn't work on bundles, so if you want to reduce the size of the filter executable, make the project with ProjectBuilder (to copy the nib, etc.), and then go into a shell and make the bundle by hand with the following commands:

```
cc -O2 -Wall -c ProgressView.m -o obj/ProgressView.o
cc -O2 -Wall -c Scale.m -o obj/Scale.o
ld -r -o Scale.bundle/Scale obj/ProgressView.o obj/Scale.o
```

Be sure to rename Scale.bundle to Scale.pfilter so IconBuilder can find it.

**Bugs in IconBuilder:**

- Filters can only be loaded before the selection tool inspector is displayed for the first time.

- By default, IconBuilder will not correctly manipulate images that are deeper than the framebuffer (despite what the page layout panel claims). To fix this, go into a shell and type:

  dwrite IconBuilder NXWindowDepthLimit TestTwenyFourBitRGB

  This allows IconBuilder to create windows and manipulate images that are twenty four bits deep, even on a mono slab. See

  /NextLibrary/Documentation/NextDev/GeneralRef/ApB_Defaults

  for more information on limiting (and increasing) window depths.

- If the Apply or Revert buttons in the selection tool inspector are clicked while there is no open document, IconBuilder will scribble in the Button's cell.

· The selection tool will not draw properly if a document is converted from one with alpha to one without alpha and the background color well is not kicked.

**Wish List for NXBitmapImageRep:**

Most of the code in the Scale object is devoted to converting the four principal bitmap formats (two bits per sample planar, eight bits per sample planar four bits per sample meshed, and eight bits per sample meshed) into a common format (eight bits per sample meshed) that the scale method can work with. It would be tremendously useful if NXBitmapImageRep had four additional methods:

- (NXColor)**readPixelAt:**(int)*i* **:**(int)*j*
- **writePixel:**(NXColor)*color* **at:**(int)*i* **:**(int)*j*
- **initData:**(unsigned *)*data*

**fromBitmap:**(NXBitmapImageRep *)*bitmap*
**pixelsWide:**(int)*width*
**pixelsHigh:**(int)*height*
**bitsPerSample:**(int)*bps*
**samplesPerPixel:**(int)*spp*
**hasAlpha:**(BOOL)*alpha*
**isPlanar:**(BOOL)*config*
**colorSpace:**(NXColorSpace)*space*
**bytesPerRow:**(int)*rowBytes*
**bitsPerPixel:**(int)*pixelBits*

- **initDataPlanes:**(unsigned char **)*planes*
**fromBitmap:**(NXBitmapImageRep *)*bitmap*
**pixels...**

These methods would provide a nice object-oriented way to manipulate bitmaps and convert them from one format to another.

Please send all bug reports to:
Linus Upson
lupson@geom.umn.edu